Regularity Problems for Visibly Pushdown Languages

Vince Bárány, Christof Loeding, Olivier Serre

Regularity Problems for Visibly Pushdown Languages

Vince Bárány[1], Christof Löding[1], and Olivier Serre[2⋆]

[1] RWTH Aachen, Germany
[2] LIAFA, Université Paris VII & CNRS, France

Abstract. Visibly pushdown automata are special pushdown automata whose stack behavior is driven by the input symbols according to a partition of the alphabet. We show that it is decidable for a given visibly pushdown automaton whether it is equivalent to a visibly counter automaton, i.e. an automaton that uses its stack only as counter. In particular, this allows to decide whether a given visibly pushdown language is a regular restriction of the set of well-matched words, meaning that the language can be accepted by a finite automaton if only well-matched words are considered as input.

1 Introduction

The class of context-free languages (CFL) plays an important role in several areas of computer science. Besides its definition using context-free grammars it has various other characterizations, the most prominent being the one via nondeterministic pushdown automata. It is well known that CFL does not enjoy good closure properties, e.g. it is not closed under complement and intersection, and that several interesting problems are undecidable, e.g. checking whether a context free language is regular, or whether it contains all words. This situation only slightly improves when considering the subclass of deterministic context free languages, i.e. languages accepted by deterministic pushdown automata (see [10] for an overview).

Another subclass of CFL that has recently been defined in [2] is the class of visibly pushdown languages. These are languages that are accepted by pushdown automata whose stack behavior (i.e. whether to execute a push, a pop, or no stack operation) is completely determined by the input symbol according to a fixed partition of the input alphabet. These automata are called visibly pushdown automata (VPA). As shown in [2, 3] this class of visibly pushdown languages enjoys many good properties similar to those of the class of regular languages, the main reason for this being that each nondeterministic VPA can be transformed into an equivalent deterministic one. Visibly pushdown automata have turned out to be useful in various context, e.g. as specification formalism for verification

⋆ Supported by the European Community Research Training Network "Games and Automata for Synthesis and Validation" (GAMES). Most of this work was done when the third author was a postdoctoral researcher at RWTH Aachen.

and synthesis problems for pushdown systems [1, 11], and as automaton model for processing XML streams [14, 12].

As each nondeterministic VPA can be determinized, all problems that concern the accepted language and that are decidable for deterministic pushdown automata are also decidable for VPAs. For example, in [15] and later with improved complexity in [16] it is shown that for a given deterministic pushdown automaton it is decidable whether its accepted language is regular. Hence, this problem is also decidable for VPAs.

In the context of validating streaming XML documents a similar question has been addressed in [14]. Phrased in the terminology of finite automata on words and trees the problem of validating streaming documents is the following: given the coding of a tree by a word using opening and closing tags around each subtree, check whether the corresponding tree belongs to a given regular tree language. It is rather simple to see that this task can be solved by a deterministic pushdown automaton that pushes a symbol onto the stack for each opening tag and pops a symbol for each closing tag. One of the questions raised and analyzed in [14] is whether one can decide for a given tree language if the streaming validation task can be solved by a finite automaton. As such an automaton has to verify that the input codes a tree, the class of these tree languages is rather restricted. The question gets more involved under the assumption that the input indeed codes a tree.

Coming back to VPAs, this assumption on the input being the coding of a tree corresponds to the assumption that the input is well-matched in the sense that each symbol that is pushed is popped eventually (each opening tag has a matching closing tag). The question of regularity of the accepted language then becomes: given a VPA, is there an equivalent finite automaton, where equivalence is restricted to the set of well-matched words? Restricting the equivalence to well-matched words can also be seen as allowing the finite automaton to count the difference between opening and closing tags to know in the end if the input was well-matched. This model is what we refer to as a visibly counter automaton (VCA). The main result of this paper is that it is decidable for a given VPA whether it is equivalent to a VCA. This problem is mentioned in [16] for deterministic pushdown automata and deterministic one-counter automata, and is to our knowledge still open.

The remainder of this paper is organized as follows. In Section 2 we provide the basic definitions of visibly pushdown and counter automata and state the main questions that we address. In Section 3 we give some basic concepts and constructions on which the decidability proofs are based. In Section 4 we show that it is decidable for a given visibly pushdown automaton whether it is equivalent to a visibly counter automaton that is allowed to test its counter value up to a certain threshold, and in Section 5 we prove that it is decidable whether such a threshold can be reduced.

We thank Victor Vianu and Luc Segoufin for drawing our attention to this topic.

2 Definitions

For a finite set X we denote the set of finite words over X by X^*. We denote by ε the empty word. For $u \in X^*$, we write $u(n)$ for the nth letter in u and $u\restriction_n$ for the prefix of length n of u, i.e., $u\restriction_0 = \varepsilon$ and $u\restriction_n = u(0)\cdots u(n-1)$ for $n \geq 1$.

A pushdown alphabet is a tuple $\widetilde{\Sigma} = \langle \Sigma_c, \Sigma_r, \Sigma_{\text{int}} \rangle$ that comprises three disjoint finite alphabets: Σ_c is a finite set of *calls*, Σ_r is a finite set of *returns*, and Σ_{int} is a finite set of *internal actions*. For any such $\widetilde{\Sigma}$, let $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_{\text{int}}$.

We define *visibly pushdown automata* over $\widetilde{\Sigma}$. Intuitively, a visibly pushdown automaton is a pushdown automaton restricted such that it pushes onto the stack only when it reads a call, it pops the stack only on reading a return, and it does not use the stack when reading an internal action.

Definition 1 (Visibly pushdown automaton [2]). *A visibly pushdown automaton (*VPA*) over $\widetilde{\Sigma}$ is a tuple $\mathcal{A} = (Q, \Sigma, \Gamma, Q_{in}, F, \Delta)$ where Q is a finite set of states, $Q_{in} \subseteq Q$ is a set of initial states, $F \subseteq Q$ is a set of final states, Γ is a finite stack alphabet, and $\Delta \subseteq (Q \times \Sigma_c \times Q \times \Gamma) \cup (Q \times \Sigma_r \times \Gamma \times Q) \cup (Q \times \Sigma_{\text{int}} \times Q)$ is the transition relation.*

To represent stacks we use a special bottom-of-stack symbol $\perp$ that is not in Γ. A *stack* is a finite sequence from the set $\perp \cdot \Gamma^*$ starting with the special symbol $\perp$ on the left, and ending with the top symbol on the right.[1] The *empty stack* is the one that only contains the symbol $\perp$.

A transition (q, a, q', γ) with $a \in \Sigma_c$ is a push-transition where on reading a, γ is pushed onto the stack and the control changes from state q to q'. Similarly, (q, a, γ, q') with $a \in \Sigma_r$ is a pop-transition where γ is read from the top of the stack and popped (if the top of stack is $\perp$, then no pop-transition can be applied), and the control state changes from q to q'. Our model (in contrast to the original definition from [2]) is therefore inherently restricted to input words having no prefix of negative stack height (to be defined below). Note that on internal actions, there is no stack operation.

A *configuration* of a VPA $\mathcal{A}$ is a pair (σ, q), where $q \in Q$ and $\sigma \in \perp \cdot \Gamma^*$. There is an *a-transition* from a configuration (σ, q) to (σ', q'), denoted $(\sigma, q) \xrightarrow{a} (\sigma', q')$ ($\mathcal{A}$ will be clear from context), if the following are satisfied.

- If a is a call, then $\sigma' = \sigma\gamma$ for some $(q, a, q', \gamma) \in \Delta$.
- If a is a return, then $\sigma = \sigma'\gamma$ for some $(q, a, \gamma, q') \in \Delta$.
- If a is an internal action, then $\sigma = \sigma'$ and $(q, a, q') \in \Delta$.

For a finite word $u = a_0 a_1 \cdots a_n$ in Σ^*, a *run* of $\mathcal{A}$ on u is a sequence of configurations $(\sigma_0, q_0)(\sigma_1, q_1)\cdots(\sigma_{n+1}, q_{n+1})$, where $q_0 \in Q_{in}$, $\sigma_0 = \perp$ and for every $0 \leq i \leq n$, $(\sigma_i, q_i) \xrightarrow{a_i} (\sigma_{i+1}, q_{i+1})$ holds. In this case we also use the notation $(\sigma_0, q_0) \xrightarrow{u} (\sigma_{n+1}, q_{n+1})$. A word $u \in \Sigma^*$ is *accepted* by a VPA if there is a run over u which ends in a final configuration, that is a configuration with

[1] Note that we are using here the reverse of the more common notation of stacks, having the top symbol on the left and the bottom on the right.

empty stack and a control state, which is final. The language $L(\mathcal{A})$ of a VPA $\mathcal{A}$ is the set of words accepted by $\mathcal{A}$.

A VPA is *deterministic* if it has a unique initial state q_{in}, and for each input letter and configuration there is at most one successor configuration. For deterministic VPAs (DVPAs) we denote the transition relation by δ instead of Δ and write $\delta(q,a) = (q',\gamma)$ instead of $(q,a,q',\gamma) \in \delta$ if $a \in \Sigma_c$, $\delta(q,a,\gamma) = q'$ instead of $(q,a,\gamma,q') \in \delta$ if $a \in \Sigma_r$, and $\delta(q,a) = q'$ instead of $(q,a,q') \in \delta$ if $a \in \Sigma_{\text{int}}$.

Let us stress, that during the run of any VPA $\mathcal{A}$ on a given word $u \in \Sigma^*$ the automaton $\mathcal{A}$ controls only which symbols are pushed on the stack, but not when a symbol is pushed or popped. At each step, the height of the stack is pre-determined by the prefix of u read thus far. Let $\chi(a)$ be the *sign* of the symbol $a \in \Sigma$ defined as $\chi(a) = 1$ if $a \in \Sigma_c$, $\chi(a) = 0$ if $a \in \Sigma_{\text{int}}$, and $\chi(a) = -1$ if $a \in \Sigma_r$. We define the *stack height* $\text{sh}(u)$ of a word $u \in \Sigma^*$ as the sum of the signs of its constituent symbols, with $\text{sh}(\varepsilon) = 0$. Furthermore, let $\text{minsh}(u) = \min\{\text{sh}(u\restriction_n) \mid 0 \le n \le |u|\}$ and $\text{maxsh}(u) = \max\{\text{sh}(u\restriction_n) \mid 0 \le n \le |u|\}$. A word u is *well matched* if $\text{sh}(u) = \text{minsh}(u) = 0$.

Given a DVPA $\mathcal{A}$ with control states Q each well-matched word $u \in \Sigma^*$ induces a transformation $T_u^{\mathcal{A}} : Q \to Q$ defined as $\{(q,q') \mid (\bot,q) \xrightarrow{u} (\bot,q')\}$, which completely describes the behavior of $\mathcal{A}$ on reading u in any context. The set of all transformations induced by a well-matched word is denoted $\mathcal{T}_{\text{wm}}^{\mathcal{A}}$. In the following we write just $\mathcal{T}_{\text{wm}}$ and T_u when $\mathcal{A}$ is understood.

The fact that VPAs control only the content of their stack but not its height allows one to determinize every VPA as shown in [2]. In the rest of the paper we will therefore assume that all VPAs considered are deterministic.

Note that we have defined acceptance with empty stack. This implies, together with the noted implicit restriction imposed by the visibility condition, that only well-matched words can be accepted. Therefore, we are considering only languages that are subsets of the language $L_{\text{wm}} = \{u \in \Sigma^* \mid \text{sh}(u) = \text{minsh}(u) = 0\}$ of well-matched words. Observe that L_{wm} is accepted by a trivial single state DVPA $\mathcal{A}_{\text{wm}}$ having a single stack symbol, hence using its stack solely as a counter to keep track of the stack height of the word being read. The following definition generalizes this concept.

Definition 2 (Visibly counter automaton). *A visibly counter automaton with threshold m (m-VCA) over $\widetilde{\Sigma}$ is a tuple $\mathcal{A} = (Q, \Sigma, q_{in}, F, \delta_0, \ldots, \delta_m)$ where Q is a finite set of states, $q_{in} \in Q$ is the initial state, $F \subseteq Q$ is a set of final states, $m \ge 0$ is a* threshold*, and $\delta_i : Q \times \Sigma \to Q$ is a transition function for every $i = 0, \ldots, m$.*

A configuration of $\mathcal{A}$ is a pair (k,q) of counter value $k \in \mathbb{N}$ and state $q \in Q$. For $a \in \Sigma$, there is an a-transition from (k,q) to (k',q'), denoted $(k,q) \xrightarrow{a} (k',q')$, if $k' = k + \chi(a)$, and $q' = \delta_k(q,a)$ if $k < m$ and $q' = \delta_m(q,a)$ if $k \ge m$.

For a finite word $u = a_0 a_1 \cdots a_n$ in Σ^*, the *run* of $\mathcal{A}$ on u is the sequence $(k_0,q_0)(k_1,q_1)\cdots(k_{n+1},q_{n+1})$ of configurations, where $q_0 = q_{in}$, $k_0 = 0$, and $(k_i,q_i) \xrightarrow{a_i} (k_{i+1},q_{i+1})$ for every $0 \le i \le n$. A word $u \in \Sigma^*$ is *accepted* by a VCA $\mathcal{A}$ if the run of $\mathcal{A}$ over u ends in a final configuration, that is a configuration

with counter value 0 and control state from F. The language $L(\mathcal{A})$ of a VCA $\mathcal{A}$ is the set of words accepted by $\mathcal{A}$.

Observe that a 0-VCA has absolutely no access to its counter, which can be perceived as an auxiliary device ensuring that only well-matched words are accepted. Other than that, a zero threshold VCA is essentially a finite automaton. Indeed, it is easy to see that a language L is accepted by some 0-VCA if and only if $L = L' \cap L_{\text{wm}}$ for some regular language L'. The next example shows that $(m+1)$-VCAs are more powerful than m-VCAs.

Example 1. Consider the languages $L_m = \{\Sigma_c^n \Sigma_r^{n-m} \Sigma_c^{l-m} \Sigma_r^l \mid m \leq l, n \in \mathbb{N}\}$ defined for each $m \in \mathbb{N}$. Each L_m consists of well-matched words and is clearly accepted by an appropriate $(m+1)$-VCA. Moreover, it is easy to show that there is no m-VCA accepting L_m.

Note that we have defined VCAs to be deterministic. In Section 5 we also use nondeterministic VCAs with the natural definition. The standard subset construction that is used to determinize finite automata can also be used to determinize VCAs.

Based on the preceding definitions we can now state the problems that we address:

(1) Given a DVPA $\mathcal{A}$ and $m \in \mathbb{N}$, is there an m-VCA that accepts $L(\mathcal{A})$?
(2) Given a DVPA $\mathcal{A}$, is there $m \in \mathbb{N}$ and an m-VCA that accepts $L(\mathcal{A})$?
(3) Given an m-VCA $\mathcal{A}$ and $m' \in \mathbb{N}$, is there an m'-VCA that accepts $L(\mathcal{A})$?

Note that decidability of the two last questions implies decidability of the first one. The following example illustrates that for (2) and (3) an exponential blow-up in the size of the automaton is unavoidable.

Example 2. Let Σ be the alphabet with $\Sigma_c = \{c_a, c_b\}$, $\Sigma_r = \{r_a, r_b\}$ and $\Sigma_{\text{int}} = \emptyset$. For a given $m \in \mathbb{N}$ let $L_m = \{c_{x_1} \cdots c_{x_m} w r_{x_m} \cdots r_{x_1} \mid x_1, \ldots, x_m \in \{a, b\} \text{ and } w \in L_{\text{wm}}\}$ be the set of well-matched words starting with m initial calls and ending with m corresponding returns. For each m, it is easily seen that L_m is accepted by a DVPA with $\mathcal{O}(m)$ states that stores the first m calls on its stack and then compares them to the m final returns. Instead of storing the initial calls on the stack it is also possible to memorize them in the control state, leading to an $(m+1)$-VCA with $\mathcal{O}(2^m)$ states. A pumping argument shows that this exponential blow-up is unavoidable.

For each m, let L'_m be the set of well-matched words that end with a sequence of m returns, where the first return in this sequence is r_a: such a language is easily accepted by an m-VCA with two states. It can also be accepted by an exponentially larger 0-VCA that remembers in its control states the last m returns. Again, a pumping argument shows that this exponential blow-up is unavoidable.

The rest of the paper is devoted to the proof of the following result (cf. Theorem 2 in Section 4 and Theorem 3 in Section 5).

Theorem 1. *Questions (1), (2) and (3) are decidable and lead to effective constructions.*

From a prior remark concerning languages accepted by 0-VCAs and from the decidability of (1) for $m = 0$ we obtain the following result.

Corollary 1. *It is decidable, whether a given* VPA $\mathcal{A}$ *accepts a regular restriction of the set of well-matched words, i.e. whether* $L(\mathcal{A}) = L \cap L_{\mathrm{wm}}$ *for some regular language* L*. When so, then a finite automaton recognizing* L *can be effectively constructed.*

Concerning the restriction that we only consider languages that are subsets of L_{wm}, note that the case where acceptance is defined only via final states can be reduced to our setting as follows. By adding a fresh symbol to Σ_r used to close unmatched calls, one can pass to a language consisting of well-matched words only. This new language can be recognized by a VCA (accepting with final states and counter value 0) iff the original language can be recognized by a VCA accepting with final states only.

3 Basic Tools and Constructions

We shall now introduce the basic concepts and tools that we are using. Throughout the rest of the paper let $\mathcal{A} = (Q, \Sigma, \Gamma, q_{in}, F, \delta)$ be a given DVPA.

We use finite single-tape and multi-tape letter-to-letter automata to represent sets and relations of configurations respectively. Therefore we assume w.l.o.g. that Q and Γ are disjoint, and identify each configuration (σ, q) of $\mathcal{A}$ with the word σq. Letter-to-letter 2-tape finite automata accept precisely the length-preserving rational relations. Basic results on length-preserving and synchronized rational relations can be found in [9]. Letter-to-letter multi-tape finite automata can be seen as classical single-tape finite automata over the product alphabet. Hence, all classical constructions and results of automata theory apply. Below we often use this fact without explicit reference. In various estimates we use the binary function $\exp(k, n)$ denoting a tower of exponentials of height k defined inductively by letting $\exp(0, n) = n$ and $\exp(k + 1, n) = 2^{\exp(k,n)}$ for all k and n.

When considering language acceptance only those configurations of $\mathcal{A}$ are of concern that are *reachable* from the initial configuration. Accordingly, in our constructions we restrict our attention to the set $V_{\mathcal{A}}$ of configurations of $\mathcal{A}$ reachable from the initial configuration. The fact, first observed by Büchi [7], that $V_{\mathcal{A}}$ is regular is therefore essential. Moreover, an obvious adaptation of the construction of [6] (see also [8]) shows that a non-deterministic finite automaton recognizing $V_{\mathcal{A}}$ with $\mathcal{O}(|Q|)$ states can be constructed in polynomial time. From now on by configuration we always mean reachable configuration, unless explicitly stated otherwise.

First we define equivalence (denoted $\sim$) of configurations of $\mathcal{A}$ in a standard way according to the languages they accept, and observe a necessary condition

(2') for a positive answer for question (2). Next we show that $\sim$, when considered as a binary relation on words describing the configurations, can be accepted by a letter-to-letter two-tape automaton. This allows us not only to decide (2') but also to prove its sufficiency.

The *configuration graph* of $\mathcal{A}$ is the edge-labelled graph $G_{\mathcal{A}} = (V_{\mathcal{A}}, E_{\mathcal{A}})$, where $V_{\mathcal{A}}$ is, as above, the set of reachable configurations of $\mathcal{A}$ and $E_{\mathcal{A}}$ is the set that contains all triples of the form $((\sigma, q), a, (\sigma', q'))$ such that $(\sigma, q), (\sigma', q') \in V_{\mathcal{A}}$, $a \in \Sigma$, and $(\sigma, q) \xrightarrow{a} (\sigma', q')$. Below we often suppress the index $\mathcal{A}$.

Definition 3 (Equivalence of configurations). *Two configurations $\sigma q, \sigma' q'$ of $\mathcal{A}$ are equivalent, in symbols $\sigma q \sim \sigma' q'$, if $|\sigma| = |\sigma'|$ and for every word $u \in \Sigma^*$ there is an accepting run of $\mathcal{A}$ labelled by u from (σ, q) to a final configuration iff there is one from (σ', q').*

Because $\mathcal{A}$ is deterministic $\sim$ is in fact a congruence with respect to the transition relations $\xrightarrow{a}$ $(a \in \Sigma)$ restricted to the set of reachable configurations. This allows us to define the quotient graph $G/_\sim$ as follows.

Definition 4 (Quotient of the configuration graph). *We define the quotient of the configuration graph $G = (V, E)$ with respect to the congruence $\sim$ as $G/_\sim = (V/_\sim, E/_\sim)$, where $V/_\sim$ consists of equivalence classes of V under $\sim$ and for all $C_1, C_2 \in V/_\sim$, and for any letter $a \in \Sigma$, $(C_1, a, C_2) \in E/_\sim$ if and only if there are some (equivalently for all) $v_1 \in C_1$ and $v_2 \in C_2$ such that $(v_1, a, v_2) \in E$.*

Note that by definition $\sigma q \sim \sigma' q'$ implies that $|\sigma| = |\sigma'|$. In other words, $\sim$ refines the equivalence defined according to stack height, i.e. $\sim$ is length-preserving. If we denote by $V|_n$ the set of reachable configurations that contain n stack symbols, i.e. $V|_n = V \cap (\bot \cdot \Gamma^n Q)$, then $\sim$ induces a certain number of equivalence classes on each set $V|_n$. In case $\mathcal{A}$ is equivalent to some m-VCA, this number of equivalence classes must be bounded by a bound independent of n, because configurations of a VCA that have the same counter value can only be distinguished by finitely many control states.

Proposition 1. *The following is necessary for (2) to have a positive answer.*

(2') *$\exists K \, \forall n$ $V|_n$ is partitioned into at most K $\sim$-equivalence classes.*

It is, however, not immediate that the above condition is also sufficient. Both to show equivalence of (2) and (2') and to prove their decidability the following observation is crucial.

Lemma 1. *One can effectively construct a letter-to-letter 2-tape automaton $\mathcal{A}_\sim$ having at most $2^{\mathcal{O}(|Q|^2)}$ states and recognizing $\sim$.*

This lemma can be shown by noting that an automaton can guess a separating word for two configurations of the same length n. Such a word consists of n returns interleaved with well-matched words. As not the particular well-matched

words u but only the transformations T_u (from the finite set $\mathcal{T}_{\mathrm{wm}}$) induced by them are interesting, a finite automaton can check whether two configurations are *not* equivalent. Then one can conclude using the closure properties of finite letter-to-letter 2-tape automata.

We are interested in the number of equivalence classes of $\sim$ for each stack height and therefore want to elect representatives for these classes. For this purpose we fix some linear ordering of the symbols of Γ and Q, thus determining the lexicographic ordering $<_{\mathrm{lex}}$ of all configurations. Note that $<_{\mathrm{lex}}$ is synchronized rational, hence, its restriction to words of equal length is recognized by a letter-to-letter automaton. Using the automata recognizing $<_{\mathrm{lex}}$, $\sim$, and V we can further construct an automaton recognizing the set $\mathrm{Rep} = \{\sigma q \in V \mid \neg\exists \sigma' q' \in V(\sigma q \sim \sigma' q' \wedge \sigma' q' <_{\mathrm{lex}} \sigma q)\}$ of lexicographically smallest representatives of each $\sim$-class as follows: One can construct a letter-to-letter automaton recognizing pairs of equivalent reachable configurations, such that the first component precedes the second one in the lexicographic ordering. After projection onto the second component, determinization, and complementation (with respect to V) one obtains a deterministic automaton $\mathcal{A}_{\mathrm{Rep}}$ recognizing Rep. The largest one of the components is the automaton $\mathcal{A}_\sim$ and the costliest operation is, of course, determinization potentially causing an exponential increase in the number of states. Thus, we obtain $\exp(2, \mathcal{O}(|Q|^2))$ as an upper bound on the size of $\mathcal{A}_{\mathrm{Rep}}$.

We now observe that (2') is equivalent to the slenderness of Rep. Following [13] and [4] we say that a language $L \subseteq \Gamma^*$ is *slender* if there is a constant K such that $|L \cap \Gamma^n| \leq K$ for all $n \in \mathbb{N}$, in which case we may also say that L is *K-thin*. Let us therefore introduce the notation $\mathrm{Rep}_n = \mathrm{Rep} \cap V|_n$. Analogously, we say that the graph $G/_\sim$ is *slender* if there is a constant K such that $|(V|_n)/_\sim| \leq K$ for all $n \geq 0$. Relying on results of [4] and [13] we immediately obtain the following.

Proposition 2. *Condition (2') is decidable, moreover, if* Rep *is K-thin, then* $K \leq |\Gamma|^{N-2} \cdot |Q| = \exp(3, \mathcal{O}(|Q|^2))$, *where N is the number of states of the minimal deterministic automaton recognizing* Rep.

Let us assume that $G/_\sim$ is slender. We identify each of its nodes C with the pair $(\mathrm{sh}(C), \mathrm{index}(C)) \in \mathbb{N} \times \{1, \cdots, K\}$, where the stack height of a class C is the stack height of any (hence all) of the configurations belonging to C and the index of C is the position of its representative $w \in C \cap \mathrm{Rep}$ with respect to $<_{\mathrm{lex}}$ among $\mathrm{Rep}_{\mathrm{sh}(C)}$. In the next section we will show that in case $G/_\sim$ is slender it is (in the above representation) actually the configuration graph of a VCA. The following lemma constitutes an important step in the proof of this result.

Lemma 2. *Assuming condition (2') holds with slenderness bound K we can effectively construct an automaton $\mathcal{C}_\sim$ reading stack contents and whose states $q_\sim$ encode mappings $\rho_{q_\sim} : Q \to \{0, \ldots, K\}$ where K is the slenderness index of $G/_\sim$. After reading a stack content σ the automaton $\mathcal{C}_\sim$ is in a state $q_\sim$ such that* $\mathrm{index}((\sigma, q)) = \rho_{q_\sim}(q)$ *for all $q \in Q$. Moreover* $\exp(5, \mathcal{O}(|Q|^2))$ *is an upper bound on the number of states of $\mathcal{C}_\sim$.*

4 From Pushdown to Counter Automata: Decidability of Question (2)

In this section we prove that slenderness is actually a sufficient condition for (2) to hold. As it is also necessary and decidable, it shows the decidability of question (2). Effectiveness follows from the proof.

Assume that $\mathcal{A}$ is a DVPA (with the usual components) such that $G_{\mathcal{A}}/_{\sim}$ is slender, and let K be a slenderness bound, i.e. there are at most K classes on each level of $G_{\mathcal{A}}/_{\sim}$.

The proof and the construction are split in two steps. First we show that $G_{\mathcal{A}}/_{\sim}$ can be effectively described by an ultimately periodic word. Then, in the second step, it easily follows that $\mathcal{A}$ is equivalent to an m-VCA with m being the offset of the ultimately periodic word.

The infinite word describing $G_{\mathcal{A}}/_{\sim}$ is such that the nth letter codes the edges of $E_{\mathcal{A}}/_{\sim}$ that leave the vertices from the nth level, i.e., the outgoing edges from the vertex set $\{(n,i) \mid i \in \{1,\dots,K\}\}$. These edges are fully described by a (partial) mapping assigning to each pair (i,a) of class index and input letter the index of the class reached from class i on level n when reading an a. If there are less than i classes on level n, then the value for (i,a) is undefined.

More formally, the description $\tau_n : \{1,\dots,K\} \times \Sigma \to \{1,\dots,K\}$ of the nth level of $G_{\mathcal{A}}/_{\sim}$ is defined by $\tau_n(i,a) = j$ iff $((n,i),a,(n+\chi(a),j)) \in E_{\mathcal{A}}/_{\sim}$ and $\tau_n(i,a)$ is undefined if (n,i) is not a vertex of $G_{\mathcal{A}}/_{\sim}$.

The sequence $\alpha := \tau_0\tau_1\dots$ completely describes $G_{\mathcal{A}}/_{\sim}$. Using the automaton $\mathcal{C}_{\sim}$ (cf. Lemma 2) it is possible to construct a finite state machine that outputs this sequence. This implies the main technical result of this section, namely that α is ultimately periodic.

Lemma 3. *The description $\alpha = \tau_0\tau_1\tau_2,\dots$ of $G_{\mathcal{A}}/_{\sim}$ is an ultimately periodic sequence that can be constructed effectively.*

As α is ultimately periodic there are numbers m and k such that $\alpha = \tau_0 \cdots \tau_{m-1}(\tau_m \cdots \tau_{m+k-1})^{\omega}$. We call m the offset and k the period of α. It is not difficult to verify that a VCA that knows whether it is in the offset part of α (using its threshold) or in the periodic part (using a modulo k counter to keep track of the position) can simulate $\mathcal{A}$. This is established in the following proposition.

Proposition 3. *If the description $\alpha = \tau_0\tau_1\dots$ of $G_{\mathcal{A}}/_{\sim}$ is ultimately periodic with offset m and period k, then one can build an m-VCA $\mathcal{B}$ such that $L(\mathcal{A}) = L(\mathcal{B})$.*

Combining Propositions 1, 2, and 3 we get the following theorem answering question (2) from Section 3.

Theorem 2. *It is decidable if for a given* VPA *there exists an equivalent* VCA*. If such a* VCA *exists it can be effectively constructed and has $\mathcal{O}((|\Gamma|\cdot|Q_{\mathcal{C}_{\sim}}|\cdot K)^{2K}\cdot K)$ states and its threshold is bounded by $\mathcal{O}((|\Gamma|\cdot|Q_{\mathcal{C}_{\sim}}|\cdot K)^{2K})$.*

5 Reducing the Threshold: Decidability of Question (3)

In all this section, we assume that $\mathcal{A} = (Q, \Sigma, q_{in}, F, \delta_0, \ldots, \delta_m)$ is an m-VCA for some threshold m. Given $m' < m$, we want to decide whether there is an m'-VCA $\mathcal{B}$ such that $L(\mathcal{A}) = L(\mathcal{B})$. If such a $\mathcal{B}$ exists, we want to provide an effective construction of it.

The decision procedure that we present consists of two steps. First, we build an m'-VCA $\mathcal{A}'$ and show that if $\mathcal{A}$ is equivalent to some m'-VCA then $L(\mathcal{A}) = L(\mathcal{A}')$. Intuitively, $\mathcal{A}'$ is a canonical candidate to be equivalent to $\mathcal{A}$. Then, we have to check whether $L(\mathcal{A}) = L(\mathcal{A}')$ holds, which is known to be decidable [2].

As the technical details of the construction of $\mathcal{A}'$ and the correctness proofs are quite involved we restrict ourselves in the following to an explanation of the underlying ideas.

The difference between $\mathcal{A}$ and an m'-VCA is that for a word w of stack height h with $m' \leq h < m$ the automaton $\mathcal{A}$ exactly knows the current stack height because it uses δ_h to compute the next configuration, whereas the m'-VCA only knows that the stack height is at least m'. Such a situation is depicted in Figure 1 (where for now we ignore all annotations except m and m').

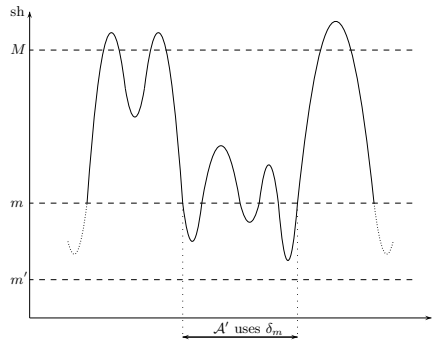

Fig. 1. A critical situation when simulating threshold m by threshold m'

The main idea is to show that, under the assumption that $\mathcal{A}$ is indeed equivalent to some m'-VCA, this additional information gained by $\mathcal{A}$ when using δ_h is not used (under certain conditions) so that instead of using δ_h to compute the next configuration one could also have used δ_m. The conditions under which it is possible to use δ_m instead of the correct transition function δ_h are also illustrated in Figure 1. If the input exceeds a certain stack height (denoted by M, a parameter depending on the size of $\mathcal{A}$), then comes back into the area between m and m', and then again goes beyond M, then one can also use δ_m when the stack height is between m and m', without changing the acceptance behavior of $\mathcal{A}$. The condition on the stack height is needed for the correctness proof to be

able to apply pumping arguments without changing the transformation on the state space that is induced by the input.

This allows to construct a nondeterministic m'-VCA $\mathcal{A}'$ that maintains in its state space a counter up to M that is updated according to the stack height. As long as the stack height stays below M, $\mathcal{A}'$ can exactly simulate $\mathcal{A}$. If the stack height exceeds M, $\mathcal{A}'$ starts using δ_m for its transitions, and it guesses the points where it can switch back to exact simulation of $\mathcal{A}$. These are the points where the stack height falls below M and reaches a value less than m' before exceeding M again. These guesses can be verified as correct by $\mathcal{A}'$ at the moment where the stack height goes below m' (because then it can compare the counter value maintained in the state space with the real stack height).

As nondeterministic VCAs can be determinized as explained in Section 2, we obtain the following lemma.

Lemma 4. *From $\mathcal{A}$ one can construct an m'-VCA $\mathcal{A}'$ such that $L(\mathcal{A}) \neq L(\mathcal{A}')$ implies that there is no m'-VCA that is equivalent to $\mathcal{A}$.*

Finally, using the fact that equivalence for VPAs (hence for VCAs) is decidable, we obtain the following result answering question (3) from Section 2.

Theorem 3. *It is decidable, given an m-VCA $\mathcal{A}$ and $m' < m$, whether $\mathcal{A}$ is equivalent to some m'-VCA, in which case such an m'-VCA $\mathcal{A}'$ can be constructed effectively.*

Concerning complexity, we note that the number of states of (the nondeterministic) $\mathcal{A}'$ is in $\mathcal{O}(|Q|^{2|Q|})$ (stemming from the definition of M). To check equivalence of $\mathcal{A}'$ with $\mathcal{A}$, one determinizes $\mathcal{A}'$ (exponential blow-up) and transforms it into a VPA: hence the complexity is doubly exponential in $|Q|$.

6 Conclusion

We have introduced the notion of visibly counter automaton as a direct adaption of standard one-counter automata to the framework of visibly pushdown automata. We have shown that it is decidable for a given VPA if it is equivalent to some VCA, even if we allow the counter to be tested up to a certain threshold, and provided an algorithm to construct such a counter automaton if it exists. This solves a special case of a problem that was posed in [16] for general deterministic pushdown automata.

A drawback of the presented proof is the high complexity of the resulting construction. The upper bound on the size of the VCA that we construct is 6-fold exponential in the size of the given visibly pushdown automaton, whereas the lower bound (Example 2) that we can prove is only singly exponential.

As a corollary of our main result we obtain that it is decidable for a given VPA whether it accepts a regular restriction of the set of well-matched words, i.e. whether its language is of the form $L \cap L_{\text{wm}}$ for a regular language L. To answer the question from [14] one would have to solve the corresponding problem

with L_{wm} replaced by another language: If we consider inputs as obtained when coding trees by words using opening and closing tags for the subtrees, then L_{wm} describes those words for which each opening tag is closed by *some* closing tag. To be a valid coding of a tree (in the sense of [14]) each opening tag has to be closed by a unique corresponding tag, i.e. the word has to be *strongly* well matched (see also [5]). Hence, to decide whether membership for a set $L(\mathcal{A})$ of coded trees can be tested by a finite automaton under the assumption that the input is well formed in the above sense, one has to check if $L(\mathcal{A})$ is of the form $L \cap L_{\text{swm}}$ for some regular language L and for L_{swm} being the set of strongly well-matched words.

Currently, we are working on the following generalization of these problems: Given two VPAs $\mathcal{A}$ and $\mathcal{B}$, is the language accepted by $\mathcal{A}$ a regular restriction of the language accepted by $\mathcal{B}$, i.e. $L(\mathcal{A}) = L \cap L(\mathcal{B})$ for some regular language L?

References

1. R. Alur, K. Etessami, and P. Madhusudan. A temporal logic of nested calls and returns. In *TACAS'04*, *LNCS* 2988, 467–481. Springer, 2004.
2. R. Alur and P. Madhusudan. Visibly pushdown languages. In *Proceedings of STOC'04*, pages 202–211. ACM, 2004.
3. R. Alur, P. Madhusudan, V. Kumar, and M. Viswanatha. Congruences for visibly pushdown languages. In *ICALP'05*, *LNCS* 3580, pages 1102–1114, 2005.
4. M. Andraşiu, G. Păun, J. Dassow, and A. Salomaa. Language-theoretic problems arising from Richelieu cryptosystems. *Theor. Comp. Sci.*, 116(2):339–357, 1993.
5. Jean Berstel and Luc Boasson. Formal properties of XML grammars and languages. *Acta Informatica*, 38(9):649–671, 2002.
6. A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR'97*, *LNCS* 1243, pages 135–150. Springer, 1997.
7. J. R. Büchi. Regular canonical systems. *Archiv für Mathematische Grundlagenforschung*, 6:91–111, 1964.
8. J. Esparza, D. Hansel, P. Rossmanith, and S Schwoon. Efficient algorithms for model checking pushdown systems. In *CAV'00*, *LNCS* 1855, pp. 232–247. Springer.
9. C. Frougny and J. Sakarovitch. Synchronized rational relations of finite and infinite words. *Theoretical Computer Science*, 108(1):45–82, 1993.
10. J. E. Hopcroft and J. D. Ullman. *Formal Languages and their Relation to Automata*. Addison-Wesley, 1969.
11. C. Löding, P. Madhusudan, and O. Serre. Visibly pushdown games. In *FST&TCS'04*, *LNCS* 3328, pages 408–420. Springer, 2004.
12. C. Pitcher. Visibly pushdown expression effects for XML stream processing. In *Programming Language Technologies for XML, PLAN-X'05*, pages 5–19, 2005.
13. G. Păun and A. Salomaa. Thin and slender languages. *Discrete Applied Mathematics*, 61(3):257–270, 1995.
14. L. Segoufin and V. Vianu. Validating streaming XML documents. In *Proceedings of PODS'02*, pages 53–64. ACM, 2002.
15. R. E. Stearns. A regularity test for pushdown machines. *Information and Control*, 11(3):323–340, 1967.
16. L. G. Valiant. Regularity and related problems for deterministic pushdown automata. *Journal of the ACM*, 22(1):1–10, 1975.

Appendix

A Proofs of Section 3

A.1 Proof of Proposition 1

Proposition 1. *The following is necessary for (2) to have a positive answer.*

(2') $\exists K\, \forall n\ V|_n$ *is partitioned into at most* K $\sim$*-equivalence classes.*

Proof. Assume $\mathcal{A}$ is equivalent to an m-VCA $\mathcal{B}$. Whenever two words u and w lead to the same unique configuration of $\mathcal{B}$, in particular $\mathrm{sh}(u) = \mathrm{sh}(w)$, then the unique configurations reached by $\mathcal{A}$ on reading u and w are equivalent, in particular having stacks of equal height. The claim follows by letting K be the number of states of $\mathcal{B}$. □

A.2 Proof of Lemma 1

Lemma 1. *One can effectively construct a letter-to-letter* 2*-tape automaton* $\mathcal{A}_\sim$ *having at most* $2^{\mathcal{O}(|Q|^2)}$ *states and recognizing* $\sim$*.*

Proof. Two configurations σq, $\sigma' q'$ are inequivalent iff they have different lengths or, since $\mathcal{A}$ is deterministic, if there is a word w accepted from precisely one of the configurations. It is sufficient to consider the equal length case.

The construction is based on the following idea: build an automaton $\mathcal{A}'_\sim$, which guesses a word w, checks both configurations for acceptance and accepts if the two results are different; finally obtain $\mathcal{A}_\sim$ after determinization and complementation (with respect to $V \times V$). It is, however, not possible to simulate the run of $\mathcal{A}$ on arbitrary words using a finite automaton. But it is also not necessary. Note that we need only consider words w leading to a configuration with an empty stack. Each such word has a unique factorization $w = w_h r_h w_{h-1} r_{h-1} \dots w_1 r_1 w_0$ with $h = |\sigma| = |\sigma'|$, such that r_i is the first return read in a configuration with stack-height i, and the w_i are well-matched words. For the configuration reached in the end the particular words w_i are not important but only the transformation $T_{w_i} \in \mathcal{T}_{\mathrm{wm}}$ induced by them. As $\mathcal{T}_{\mathrm{wm}}$ is finite, $\mathcal{A}'_\sim$ may now proceed reading words from right to left as follows. It first reads the states q and q' and maintains two states throughout the simulation. In each step $\mathcal{A}'_\sim$ guesses a transformation $T \in \mathcal{T}_{\mathrm{wm}}$ and a return symbol r, reads the next top stack symbols of both configurations and simulates the two runs accordingly. From $\mathcal{A}'_\sim$ one easily constructs an equivalent non-deterministic automaton having the same states reading words from left to right and with exchanged roles of initial and final states. Finally, after determinization and complementation (with respect to $V \times V$) we do obtain $\mathcal{A}_\sim$ as promised. □

A.3 Proof of Proposition 2

Proposition 2 *Condition (2') is decidable, moreover, if* Rep *is K-thin, then $K \leq |\Gamma|^{N-2} \cdot |Q| = \exp(3, \mathcal{O}(|Q|^2))$, where N is the number of states of the minimal deterministic automaton recognizing* Rep.

Proof. This problem has been considered and shown to be decidable in [4] and refined in [13]. In these papers the equivalence of the following properties of a regular language L is shown.

(i) L is slender;
(ii) L is a finite union of single loops, i.e. $L = \bigcup_{i=1}^{n} u_i v_i^* w_i$ for some words u_i, v_i, w_i $(i = 1, \ldots, n)$;
(iii) the minimal deterministic automaton A recognizing L has the *unique loop property*, meaning that for each reachable state s of A there are only finitely many paths from the initial state to s, and from s to a final state, which do not contain s as an intermediate node; and further, all words taking A from s to s are powers of a unique nonempty word.

Condition (iii) is obviously decidable. From (ii) one immediately derives the upper bound $|\Gamma|^{N-1}$ on the thinness of L, where Γ is the alphabet of L and N is the number of states of the minimal automaton recognizing L. Indeed, if L is K-thin, then $K \leq n$ (where n is as in (ii) above) and any automaton accepting L must have distinct simple accepting paths labelled by $u_i w_i$ for each $i = 1, \ldots, n$ and evidently there can be no more than $|\Gamma|^{N-1}$ distinct simple paths on N states. This bound is tight as witnessed for instance by the minimal deterministic automaton recognizing $\Gamma^N a^*$ with $a \in \Gamma$. Note that for Rep we have to adjust the upper bound to $|\Gamma|^{N-2} \cdot |Q|$, since the last symbol on any accepting path is a state of $\mathcal{A}$. □

A.4 Proof of Lemma 2

Lemma 2. *Assuming condition (2') holds with slenderness bound K we can effectively construct an automaton $\mathcal{C}_\sim$ reading words over Γ and whose states $q_\sim$ encode mappings $\rho_{q_\sim} : Q \to \{0, \ldots, K\}$ where K is the slenderness index of $G/_\sim$. After reading a stack content σ the automaton $\mathcal{C}_\sim$ is in a state $q_\sim$ such that* $\mathrm{index}((\sigma, q)) = \rho_{q_\sim}(q)$ *for all $q \in Q$. Moreover* $\exp(5, \mathcal{O}(|Q|^2))$ *is an upper bound on the number of states of $\mathcal{C}_\sim$.*

Proof. Assume Rep to be K-thin. We claim that for each $i = 1, \ldots, K$ there is an automaton $\mathcal{A}^{(i)}$ recognizing the set $V^{(i)}$ of reachable configurations belonging to classes of index at most i. Then $V^{(K)}$ is just the set of reachable states, and for $i < K$, $V^{(i)}$ is the set of $w \in V$ satisfying the following formula.

$$\neg \exists w_0 \ldots \exists w_i \Big(\bigwedge_{j=0}^{i} (w_j \in \mathrm{Rep} \wedge |w_j| = |w|) \wedge w_0 \sim w \wedge \bigwedge_{j=0}^{i-1} w_{j+1} <_{\mathrm{lex}} w_j \Big)$$

Hence, we can construct $\mathcal{A}^{(i)}$ accordingly, by first constructing an $(i+2)$-tape letter-to-letter automaton recognizing tuples $(w, w_0, \ldots, w_i)$ of configurations satisfying the matrix of the above formula, then projecting the transitions onto the first component, finally, complementing (with respect to V) the automaton obtained after determinization.

For each $i = 1, \ldots, K$ let $\mathcal{A}^{(i)} = (Q^{(i)}, {q_0}^{(i)}, \delta^{(i)}, F^{(i)})$ be the automaton thus obtained. We define $\mathcal{C}_\sim$ to be the direct product of all $\mathcal{A}^{(i)}$ $(i = 1, \ldots, K)$. The states of $\mathcal{C}_\sim$ are thus tuples $q_\sim = (q^{(1)}, \ldots, q^{(K)})$ of states of the $\mathcal{A}^{(i)}$'s. Let $q_\sim = (q^{(1)}, \ldots, q^{(K)})$ be the state reached after reading some stack content σ. Since the sets $V^{(i)} \setminus V^{(i-1)}$ partition the set of reachable configurations we can associate a mapping $\rho_{q_\sim} : Q \to \{0, \ldots, K\}$ to $q_\sim$ as follows.

$$\rho_{q_\sim}(q) = \begin{cases} \min i & \text{such that } \delta^{(i)}(q^{(i)}, q) \in F^{(i)} \\ 0 & \text{if there is no such } i \end{cases}$$

This definition corresponds to the following observation: σq is reachable iff there is an $\mathcal{A}^{(i)}$ accepting it, in which case the index of the equivalence class of σq is the least such i.

Looking at the size of the automata we obtain the following upper bounds. The construction of $\mathcal{A}^{(i)}$ involves the product of i copies of $\mathcal{A}_{\text{Rep}}$ each of size $\exp(2, \mathcal{O}(|Q|^2))$, where i itself can be as large as $K - 1 = \exp(3, \mathcal{O}(|Q|^2))$. The number of these product states alone is $\exp(4, \mathcal{O}(|Q|^2))$ and the factor corresponding to the additional components responsible for checking $\sim$, $<_{\text{lex}}$, etc. is negligible in comparison. After determinization we have to calculate with an additional exponential blowup, giving the final bound of $\exp(5, \mathcal{O}(|Q|^2))$ on the size of the $\mathcal{A}^{(i)}$. Altogether, we obtain $\exp(5, \mathcal{O}(|Q|^2))$ as an upper bound on the number of states of $\mathcal{C}_\sim$ as well. □

B Details for Section 4

For simplicity, we identify the states from $\mathcal{C}_\sim$ with the mappings encoded by them (cf. Lemma 2).

We aim at showing that the sequence $\tau_0, \tau_1, \ldots$ is ultimately periodic by constructing a finite state transducer without input that produces this sequence as output. The main ingredient for constructing this transducer is the automaton $\mathcal{C}_\sim$. If the transducer should output τ_n it has to be able to compute the values $\tau_n(i, a)$. Assume that (σ, q) is the representative of class i on level n and that the transducer knows the state ρ of $\mathcal{C}_\sim$ reached after reading σ. If a is an internal action or a call, one can directly derive the value $\tau_n(i, a)$ from $\delta(q, a)$:

- If $a \in \Sigma_{\text{int}}$ and $\delta(q, a) = q'$, then $\tau_n(i, a) = \rho(q')$.
- If $a \in \Sigma_c$ and $\delta(q, a) = (q', \gamma)$, then $\tau_n(i, a) = \rho'(q')$, where $\rho' = \delta_{\mathcal{C}_\sim}(\rho, \gamma)$.

If a is a return, γ is the topmost symbol of σ, i.e., $\sigma = \sigma'\gamma$, and if $\delta(q, \gamma, a) = q'$, then the transducer needs the state ρ' reached by $\mathcal{C}_\sim$ after reading σ'. In this case $\tau(i, a) = \rho'(q')$. To be able to implement this, the transducer has to simulate $\mathcal{C}_\sim$

on the representatives of the different classes for each level. The following lemma states that a representative of some class in level n can always be constructed from a representative of some class in level $n-1$.

Lemma B1 *For all $n \geq 1$ and for all $(\sigma, q) \in \mathrm{Rep}_n$ with $\sigma = \sigma'\gamma$ for some $\gamma \in \Gamma$ there exists $q' \in Q$ such that $(\sigma', q') \in \mathrm{Rep}_{n-1}$.*

Proof. Let $\sigma = \sigma'\gamma$. Let $q'' \in Q$ be such that $\mathrm{index}((\sigma', q''))$ is minimal with the constraint that there is a word $u \in L_{\mathrm{wm}} \cdot \Sigma_c \cdot L_{\mathrm{wm}}$ such that $(\sigma', q'') \xrightarrow{u} (\sigma, q)$. Note that such a q'' always exists as (σ, q) is reachable. There is some $(\sigma'', q') \in \mathrm{Rep}_{n-1}$ such that $(\sigma', q'') \sim (\sigma'', q')$. If $(\sigma''\gamma'', q''')$ denotes the configuration such that $(\sigma'', q') \xrightarrow{u} (\sigma''\gamma'', q''')$, then $(\sigma''\gamma'', q''') \sim (\sigma, q)$. As $(\sigma, q) \in \mathrm{Rep}$ we know that $\sigma \leq_{lex} \sigma''\gamma''$ and thus $\sigma' \leq_{lex} \sigma''$. Furthermore, $\sigma'' \leq_{lex} \sigma'$ since $(\sigma'', q') \in \mathrm{Rep}$. We conclude that $\sigma' = \sigma''$ and thus $(\sigma', q') \in \mathrm{Rep}_{n-1}$. □

Using this we can show that a finite transducer can indeed maintain the information needed to output the description of $G_{\mathcal{A}}/_{\sim}$.

Lemma 3. *The description $\alpha = \tau_0\tau_1\tau_2, \ldots$ of $G_{\mathcal{A}}/_{\sim}$ is an ultimately periodic sequence that can be constructed effectively.*

Proof. Throughout this proof we assume for simplicity that each level has exactly K classes. The general case only requires the use of an additional symbol indicating that the corresponding class does not exist on this level.

For $n \in \mathbb{N}$ and $i \in \{1, \ldots, K\}$ let σ_i^n denote the stack content of the representative of class i on level n. From Lemma B1 it follows that there are some $j \in \{1, \ldots, K\}$ and some $\gamma_i^n \in \Gamma$ such that $\sigma_i^n = \sigma_j^{n-1}\gamma_i^n$. If j_i^n denotes the minimal such j, then σ_i^n can be coded by the pair (j_i^n, γ_i^n). If we further enrich this information with the state ρ_i^n of $\mathcal{C}_{\sim}$ reached after reading σ_i^n, we have all the information needed to produce the description of $G_{\mathcal{A}}/_{\sim}$.

First note, that from $(j_1^{n-1}, \gamma_1^{n-1}, \rho_1^{n-1}), \ldots, (j_K^{n-1}, \gamma_K^{n-1}, \rho_K^{n-1})$ one can easily construct $(j_i^n, \gamma_i^n, \rho_i^n)$ for each $i \in \{1, \ldots, K\}$:

- For each $\ell \in \{1, \ldots, K\}$ (in ascending order) pick a state q such that $\rho_\ell^{n-1}(q) = \ell$. Such a state must exists because ρ_ℓ^{n-1} is the state of $\mathcal{C}_{\sim}$ reached after processing σ_ℓ^{n-1}, the stack content of the representative of class ℓ on level $n-1$.
- For each $\gamma \in \Gamma$ in lexicographically ascending order compute ρ_γ, the successor state of ρ_ℓ^{n-1} in $\mathcal{C}_{\sim}$ when reading γ. If $\rho_\gamma(q') = i$ for some $q' \in Q$, then $\gamma_i^n = \gamma$, $j_i^n = \ell$, and $\rho_i^n = \rho_\gamma$.

This allows to construct a transducer that starts in state $(1, \bot, \rho_1^0), \ldots, (1, \bot, \rho_K^0)$ and after having made n steps is in state

$$\begin{bmatrix} (j_1^n, \gamma_1^n, \rho_1^n), \ldots, (j_K^n, \gamma_K^n, \rho_K^n) \\ (j_1^{n-1}, \gamma_1^{n-1}, \rho_1^{n-1}), \ldots, (j_K^{n-1}, \gamma_K^{n-1}, \rho_K^{n-1}) \end{bmatrix}$$

It remains to show that from this information the description τ_n of the nth level of $G_{\mathcal{A}}/_{\sim}$ can be constructed. The idea for this was already explained at the beginning of this section.

To define $\tau_n(i,a)$ for $i \in \{1,\dots,K\}$ and $a \in \Sigma$ we make the usual case distinction.

- If $a \in \Sigma_c$ and $\delta(q,a) = (q',\gamma)$, then $\tau_n(i,a) = \rho'(q')$ for $\rho' = \delta_{\mathcal{C}_\sim}(\rho_i^n, \gamma)$.
- If $a \in \Sigma_{\text{int}}$ and $\delta(q,a) = q'$, then $\tau_n(i,a) = \rho_i^n(q')$.
- If $a \in \Sigma_r$ and $\delta(q,a,\gamma_i^n) = q'$, then $\tau_n(i,a) = \rho_{j_i^n}^{n-1}(q')$.

For the last item note that popping γ_i^n from σ_i^n results in the stack content $\sigma_{j_i^n}^{n-1}$ as j_i^n indicates the class of level $n-1$ from which σ_i^n is constructed by appending γ_i^n. □

Proposition 3. *If the description $\alpha = \tau_0\tau_1\dots$ of $G_{\mathcal{A}}/_\sim$ is ultimately periodic with offset m and period k, then one can build an m-VCA $\mathcal{B}$ such that $L(\mathcal{A}) = L(\mathcal{B})$.*

Proof. To simulate $\mathcal{A}$ it is sufficient to always know the current vertex of $G_{\mathcal{A}}/_\sim$ and all the required information is coded in $\alpha = \tau_0 \cdots \tau_{m-1}(\tau_m \cdots \tau_{m+k-1})^\omega$. The threshold m of $\mathcal{B}$ ensures that we always know whether we are in the offset part of α or in the periodic part. If we are in the periodic part we need to remember a number from 0 to $k-1$ to locate the exact position in the periodic part. This information is easily maintained in the state space by a modulo k counter. Furthermore, we need to know the current class that we are in. This information can be updated using the functions τ_i.

The formal definition of $\mathcal{B} = (Q_{\mathcal{B}}, \Sigma, q_{\mathcal{B}}^{\text{in}}, F_{\mathcal{B}}, \delta_0, \dots, \delta_m)$ is as follows:

- $Q_{\mathcal{B}} = \{1,\dots,K\} \times \{0,\dots,k-1\}$.
- $q_{\mathcal{B}}^{\text{in}} = (\text{index}((\bot, q_{\mathcal{A}}^{\text{in}})), 0)$.
- $F_{\mathcal{B}} = \{\text{index}((\bot,q)) \mid q \in F_{\mathcal{A}}\} \times \{0,\dots,k-1\}$.
- For every $j < m$, $i \in \{1,\dots,K\}$, $r < k$, and $a \in \Sigma$:

$$\delta_j((i,r),a) = (\tau_j(i,a), 0).$$

- For every $r < k$, $i \in \{1,\dots,K\}$, and $a \in \Sigma$:

$$\delta_m((i,r),a) = (\tau_{r+m}(i,a), (r + \chi(a)) \mod k).$$

For the definition of the final states note that the only possibility to reach a final state (q,r) with $r \neq 0$ after reading a well-matched word is for $m = 0$ and $r = k-1$. For the definition of δ_j with $j < m$ note that the only configurations $((q,r),j)$ with $r \neq 0$ that can be reached are those with $j = m-1$ and $r = k-1$. □

C Details for Section 5

Here, we give the technical details for the proof of Lemma 4.

Recall that $\mathcal{T}_{\text{wm}}$ denotes the set of transformations induced on the state space of a VPA by well-matched words. This definition is also valid for VCAs, where the

transformation T_w for $w \in \mathcal{T}_{\mathrm{wm}}$ is now defined by $T_w(q) = q'$ if $(0,q) \xrightarrow{w} (0,q')$, i.e. the empty stack is replaced by counter value 0. We call T_w accepting if $T_w(q_{in}) \in F$. Obviously, T_w is accepting if and only if $w \in L(\mathcal{A})$.

One can also obtain T_w as the concatenation of transformation by factors of w. If these factors are not well matched one has to take into account the stack height of the word before the current factor. Formally, we define for every word w' and every integer $h \geq -\mathrm{minsh}(w')$ a transformation $T^h_{w'} : Q \to Q$ by setting $T^h_{w'}(q) = q'$ if and only if w' leads in $\mathcal{A}$ from (h,q) to $(h + \mathrm{sh}(w'), q')$. Now consider any factorization $w_1 \cdots w_k$ of w. Then $T_w = T^{h_k}_{w_k} \circ \cdots \circ T^{h_1}_{w_1}$ where $h_i = \mathrm{sh}(w_1 \cdots w_{i-1})$ for every $i = 1, \ldots, k$. The property $h_i \geq -\mathrm{minsh}(w_i)$ follows from $w \in L_{\mathrm{wm}}$.

For $M = (|Q|^{|Q|})^2 + 1 + m$ we define the following languages on Σ.

- $U = \{u \in \Sigma^* \mid u \in L_{\mathrm{wm}} \text{ and } \mathrm{maxsh}(u) \geq M - m\}$.
- $X = \{u \in \Sigma^* \mid \mathrm{sh}(u) = 0 \text{ and } 0 > \mathrm{minsh}(u) \geq m' - m\}$.

For the choice of M note that $|Q|^{|Q|}$ is the number of transformations on Q. If the stack height of some well-matched word exceeds M, then this allows us to pump this word to reach arbitrary stack heights without changing the transformation induced by this word.

Lemma C1 *For every word $u \in U$ and every integer $h > \mathrm{maxsh}(u)$ there exists a word $u' \in U$ such that $\mathrm{maxsh}(u') \geq h$ and $T^m_u = T^m_{u'}$.*

Proof. This follows from $\mathrm{maxsh}(u) \geq (|Q|^{|Q|})^2 + 1$ by using a pumping argument. More precisely, a simple counting argument shows that there are words $u_1, \ldots, u_5$ with $\mathrm{sh}(u_3) = 0$, $\mathrm{sh}(u_2) > 0$, and $\mathrm{sh}(u_2) = -\mathrm{sh}(u_4)$ such that $u = u_1u_2u_3u_4u_5$, $T^m_{u_1} = T^m_{u_1u_2}$, and $T^m_{u_1u_2u_3} = T^m_{u_1u_2u_3u_4}$. See Figure 2 for an illustration. Therefore, $T^m_u = T^m_{u_1u_2^\ell u_3 u_4^\ell u_5}$ for every $\ell \geq 0$. □

Let $w = w_1 \cdots w_k$. From now on, we implicitly assume that w is associated with this decomposition. A factor w_i in this factorization is a *special factor* if and only if $\mathrm{sh}(w_1 \cdots w_{i-1}) = m$ and $w_i \in UXU$. See Figure 3 for an illustration.

We associate with w another transformation $T^*_w = T^+_{w_k} \circ \cdots \circ T^+_{w_1}$ where $T^+_{w_i}$ is defined as follows:

- If w_i is not special then $T^+_{w_i} = T^{h_i}_{w_i}$ with $h_i = \mathrm{sh}(w_1 \cdots w_{i-1})$.
- If w_i is special then $T^+_{w_i}$ is the transformation defined by $T^+_{w_i}(q) = q'$ for $q' := \delta_m(q, w_i)$. Here, δ_m applied to a word instead of a letter is defined as usual by $\delta_m(q, \varepsilon) = q$ and $\delta_m(q, ua) = \delta_m(\delta_m(q,u), a)$.

This definition corresponds to the idea that is described in Section 5. Inside a special factor only δ_m is used to compute the transformation on the state space, even if the stack height is between m and m'.

We have the following necessary condition for $\mathcal{A}$ to be equivalent to some m'-VCA.

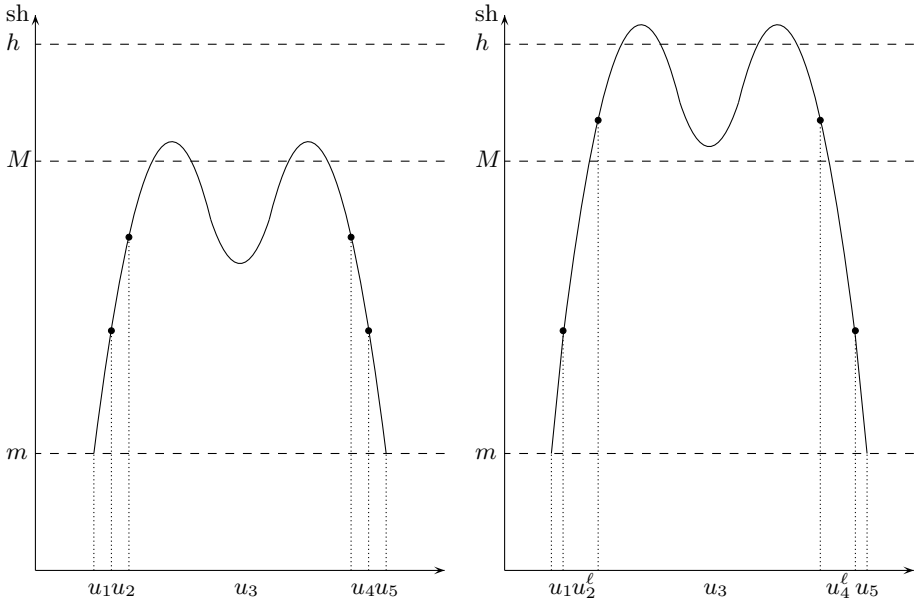

Fig. 2. Lemma C1: from u to u'

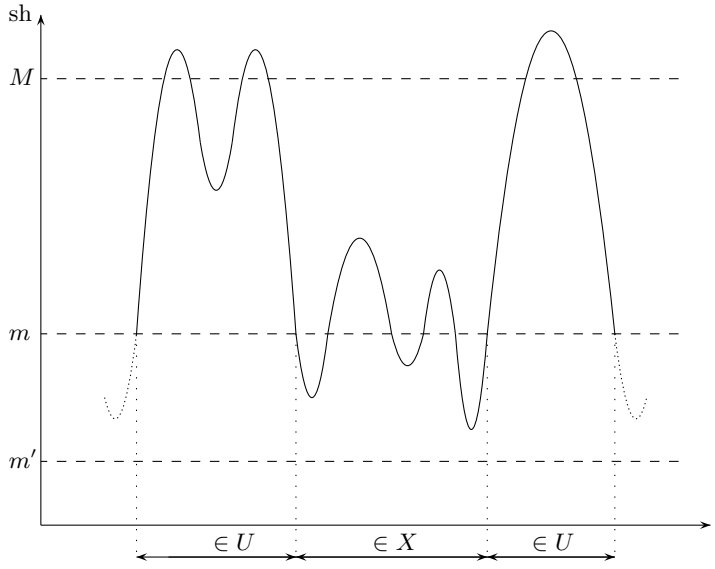

Fig. 3. A special factor

Proposition C2 *If $\mathcal{A}$ is equivalent to some m'-*VCA*, then for every word $w \in L_{\mathrm{wm}}$ and every factorization $w_1 \cdots w_k$ of it, T_w is accepting if and only if T^*_w is accepting.*

Proof. Assume that $\mathcal{A}$ is equivalent to some m'-VCA $\mathcal{B}$. We prove the result by induction on the number of special factors appearing in the decomposition of w. If there is no such factor, the result holds as in this case $T_w = T^*_w$.

Assume that the result is proved for words involving at most s special factors for some $s \geq 0$. Let w be a word with $s+1$ special factors, and let $w_1 \dots w_k$ be the underlying factorization. Let w_i be some special factor.

In the sequel we construct a word $w'' = w''_1 \dots w''_k$ that is in L_{wm} such that:

1. for all $j \neq i$ we have $w''_j = w_j$,
2. w''_i is not special,
3. $T^*_{w''} = T^*_w$,
4. $w'' \in L(\mathcal{A})$ iff $w \in L(\mathcal{A})$.

These properties allow us to conclude. The two first conditions imply that there are s special factors in the factorization of w''. This allows us to apply the induction hypothesis to w''. From the fourth condition we obtain that T_w is accepting if and only if $T_{w''}$ is. The induction hypothesis yields that $T_{w''}$ is accepting if and only if $T^*_{w''}$ is. Finally, the third property implies that $T^*_{w''}$ is accepting if and only if T^*_w is.

Let us now explain how to build w'', that is how to build w''_i (note that due to the first property no choice is allowed for the other factors). As $w_i \in UXU$, there are words $u, v \in U$ and $x \in X$ such that $w_i = uxv$.

Let $\mathcal{B} = (Q^{\mathcal{B}}, \Sigma, q^{\mathcal{B}}_{in}, F^{\mathcal{B}}, \delta^{\mathcal{B}}_0, \dots, \delta^{\mathcal{B}}_{m'})$ be an m'-VCA for $L(\mathcal{A})$ and let $n = |Q^{\mathcal{B}}|$. Our objective is to be able to manipulate u and v in such a way that the transformations induced on Q and on $Q^{\mathcal{B}}$ are not affected. For this purpose, we first have to increase the maximal stack height of u and v.

We apply Lemma C1 to u and v with $h = (m - m')[(n^n|Q|^{|Q|})^2] + 1$. We obtain words u' and v' with $\mathrm{maxsh}(u), \mathrm{maxsh}(v) \geq h$ and $T^m_{u'} = T^m_u$, $T^m_{v'} = T^m_v$. For $w'_i = u'xv'$ this directly implies $T^m_{w_i} = T^m_{w'_i}$ as well as $T^+_{w_i} = T^+_{w'_i}$. Moreover, let $w' = w'_1 \dots w'_k$ where $w'_j = w_j$ for every $j \neq i$. From the previous observations we get $T^*_w = T^*_{w'}$ and $T_w = T_{w'}$, where the latter equality also implies that $w' \in L(\mathcal{A})$ if and only if $w \in L(\mathcal{A})$.

To obtain w''_i from w'_i we "lift" the middle part x of w'_i such that the stack height does not go below m anymore. This lifting has to be done in such a way that the transformation induced by the new word is accepting if and only if the transformation induced by the former word is. We again apply a pumping argument but now in such a way that the behavior of both, $\mathcal{A}$ and $\mathcal{B}$, are not affected. This is the reason why we first increased the stack height of u and v.

For every word y with $\mathrm{minsh}(y) \geq m' - m$ we consider the transformations $T^{\mathcal{B}}_y$ on $Q^{\mathcal{B}}$ defined by $T^{\mathcal{B}}_y(q) = \delta^{\mathcal{B}}_{m'}(q, y)$ for every $q \in Q^{\mathcal{B}}$.

As $\mathrm{sh}(u'), \mathrm{sh}(v') \geq h$, one can use a counting argument to come up with two factorizations $u' = u'_1u'_2u'_3$ and $v' = v'_1v'_2v'_3$ that have the following properties (see Figure 4 for an illustration):

(a) $\mathrm{sh}(u'_1) = -\mathrm{sh}(v'_3)$, $\mathrm{sh}(u'_2) = -\mathrm{sh}(v'_2)$, and $\mathrm{sh}(u'_3) = -\mathrm{sh}(v'_1)$.
(b) $\mathrm{sh}(v'_2) \geq m - m'$.
(c) $T^{\mathcal{B}}_{u'_1} = T^{\mathcal{B}}_{u'_1 u'_2}$ and $T^{\mathcal{B}}_{v'_3} = T^{\mathcal{B}}_{v'_2 v'_3}$,
(d) $T^{m}_{u'_1} = T^{m}_{u'_1 u'_2}$ and $T^{m}_{v'_3} = T^{m}_{v'_2 v'_3}$.

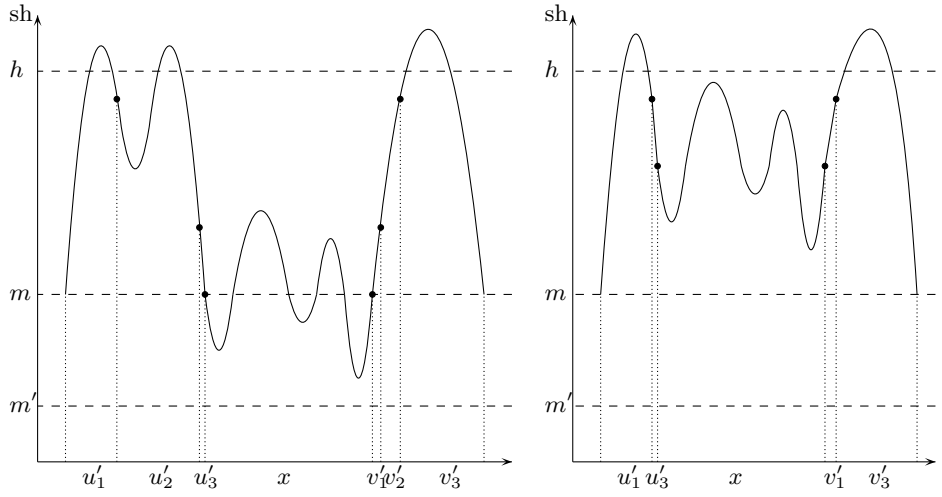

Fig. 4. From w'_i to w''_i

We set $u'' = u'_1 u'_3$, $v'' = v'_1 v'_3$ and $w''_i = u'' x v''$. From (a) we get that w''_i is well matched and from (b) that it is not a special factor anymore. Furthermore, according to (c), cutting out u'_2 and v'_2 does not change the transformation induced on $\mathcal{B}$. Hence, $T^{\mathcal{B}}_{w'_i} = T^{\mathcal{B}}_{w''_i}$ and thus $w' \in L(\mathcal{B})$ if and only if $w'' \in L(\mathcal{B})$. As $L(\mathcal{A}) = L(\mathcal{B})$ we get $w'' \in L(\mathcal{A})$ if and only if $w' \in L(\mathcal{A})$. We have already seen that $w' \in L(\mathcal{A})$ if and only if $w \in L(\mathcal{A})$ and therefore we obtain the fourth property required for w''.

It remains to prove the third property. For this note that (d) implies that $T^{+}_{w'_i} = T^{m}_{w''_i}$. As w''_i is not a special factor we obtain $T^{m}_{w''_i} = T^{+}_{w''_i}$. Therefore, $T^{*}_{w''} = T^{*}_{w'} = T^{*}_{w}$. □

According to Proposition C2, if $\mathcal{A}$ is equivalent to some m'-VCA, it is sufficient to check whether $T^{*}_{w}(q_{\mathrm{in}}) \in F$ to decide whether a word w is in $L(\mathcal{A})$. Note that *any* factorization of w can be considered. We show that there is a nondeterministic m'-VCA $\mathcal{A}' = (Q', \Sigma, q'_{\mathrm{in}}, F', \delta'_0, \ldots, \delta'_{m'})$ that can compute $T^{*}_{w}(q_{\mathrm{in}})$ for some factorization that it guesses.

In the following we describe how $\mathcal{A}'$ works. As we describe a nondeterministic VCA there might be runs that cannot be continued because no matching transition is defined. Complete runs are called *successful*. Note that with this

terminology "successful" does not say anything about the run being accepting or not.

The m'-VCA $\mathcal{A}'$ has two modes: exact (EX) and approximate (AP). In the exact mode it mimics $\mathcal{A}$ while in the approximate mode it mimics δ_m. The mode is encoded in the control state. Changing the mode is done nondeterministically and corresponds to guessing the factorization that determines T^*_w.

The set of control states is $Q' = Q \times \{\text{AP}, \text{EX}\} \times \{?, !\} \times \{0, \ldots, M\}$. The last component is for a counter κ that counts sh up to M. This counter is implemented in such a way that in each successful run of $\mathcal{A}'$, whenever the mode is EX, then κ corresponds exactly to the stack height. This is in particular used when being in the exact mode with stack height between m' and m.

For each $j \leq m'$, δ'_j mimics δ_j and updates κ in the natural way. To ensure that the value of κ is correct we can only apply δ'_j to states where the value of κ equals j. Furthermore, we have to be in exact mode if κ is smaller than m'.

The behavior of $\delta'_{m'}$ depends on the current mode. In exact mode it mimics $\delta_{\min(m,\kappa)}$, and when being in the approximate mode it mimics δ_m.

The mode can only be changed if $\kappa = m$. The positions where the mode changes define the factorization of w according to which T^*_w is computed by $\mathcal{A}'$. We construct $\mathcal{A}'$ in such a way that the factors w_i corresponding to the approximate mode are either from U or from UXU. In both cases using δ_m computes $T^+_{w_i}$. If we can ensure these properties, then $\mathcal{A}'$ computes indeed T^*_w for the factorization induced by the mode switches. To implement the mode switches correctly we use the third component of the control state that is either ? or !, where ? means that it is not yet verified that the current mode is correct and ! means that we are in the correct mode. Several rules are used to implement this verification.

The rules for this verification are the following (see Figure 5 for an illustration):

- If we are in EX and ?, then the counter value κ is not allowed to reach M and furthermore it has to reach a value smaller than m' at some point. Then it can be verified by $\mathcal{A}'$ that the value of κ is correct and we can switch to EX with !.
- If we are in EX and !, then the counter value κ is not allowed to reach M (as before). If κ equals m we can nondeterministically choose to switch to AP with ?.
- If we are in AP with ?, then the counter value κ is not allowed to reach values smaller than or equal to m. We switch to AP with ! once κ equals M.
- If we are in AP with !, then we can either leave $\kappa = M$ unchanged or we start counting down when reading a return. Once we started changing κ it is not allowed to reach M again. If κ equals m we switch to EX with ?.

In this way we construct $\mathcal{A}'$ in such a way that each successful run on input w has the following properties (that have already been mentioned above):

1. If the mode of a state is EX, then the counter value κ of the state equals the counter value of the configuration (i.e., the stack height).

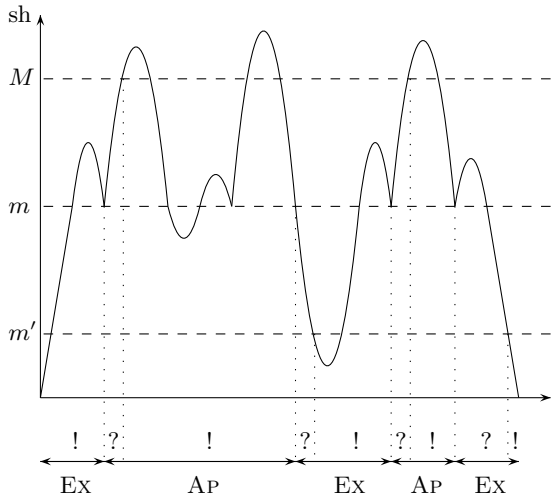

Fig. 5. Modes along the input

2. If we pick a maximal subword of w that corresponds to a sequence of states in approximate mode, then this subword is either in U or in UXU.

With these properties and the way the transitions are defined it is clear that $\mathcal{A}'$ computes $T^*_w(q_{\text{in}})$ for the factorization induced by the mode switches. This leads to the following result (already stated in Section 5).

Lemma 4. *From $\mathcal{A}$ one can effectively construct an m'-*Vca *$\mathcal{A}'$ such that $L(\mathcal{A}) \neq L(\mathcal{A}')$ implies that there is no m'-*Vca *that is equivalent to $\mathcal{A}$.*

Proof. We give a formal definition of the Vca $\mathcal{A}' = (Q', \Sigma, q'_{in}, F', \delta'_0, \ldots, \delta'_{m'})$ that we described above. Proposition C2 allows us to conclude $L(\mathcal{A}) = L(\mathcal{A}')$ and as mentioned in Section 2 $\mathcal{A}'$ can be determinized.

We define $\mathcal{A}'$ formally as follows:

- $Q' = Q \times \{\text{Ap}, \text{Ex}\} \times \{?, !\} \times \{0, \ldots, M\}$.
- $q'_{\text{in}} = (q_{\text{in}}, \text{Ex}, !, 0)$.
- $F' = \{(q, \text{Ex}, !, 0) \mid q \in F\}$. If $m' = 0$ then we also add $\{(q, \text{Ex}, ?, 0) \mid q \in F\}$ to the final states because in this case $\mathcal{A}'$ cannot switch back to Ex with ! as no counter values below m' can be reached.
- For $j < m'$, δ'_j contains the following transitions for all $a \in \Sigma$ and $q \in Q$:

$$(q, \text{Ex}, ?, j) \xrightarrow{a} (\delta_j(q, a), \text{Ex}, !, j + \chi(a))$$
$$(q, \text{Ex}, !, j) \xrightarrow{a} (\delta_j(q, a), \text{Ex}, !, j + \chi(a))$$

– The transition relation $\delta'_{m'}$ contains the following transitions for all $q \in Q$ and $a \in \Sigma$, where $q' := \delta_m(q, a)$:

$$
\begin{array}{rll}
(q, \textsc{Ex}, !, \kappa) \xrightarrow{a} (q'', \textsc{Ex}, !, \kappa + \chi(a)) & & q'' = \delta_{\min(\kappa,m)}(q, a),\ m' \leq \kappa < M \\
(q, \textsc{Ex}, !, m) \xrightarrow{a} (q', \textsc{Ap}, ?, m + \chi(a)) & & \\
(q, \textsc{Ap}, ?, \kappa) \xrightarrow{a} (q', \textsc{Ap}, ?, \kappa + \chi(a)) & & m < \kappa < M \\
(q, \textsc{Ap}, ?, \kappa) \xrightarrow{a} (q', \textsc{Ap}, !, \kappa + \chi(a)) & & \kappa + \chi(a) = M,\ m < \kappa < M \\
(q, \textsc{Ap}, !, M) \xrightarrow{a} (q', \textsc{Ap}, !, M) & & \\
(q, \textsc{Ap}, !, M) \xrightarrow{a} (q', \textsc{Ap}, !, M - 1) & & a \in \Sigma_r \\
(q, \textsc{Ap}, !, \kappa) \xrightarrow{a} (q', \textsc{Ap}, !, \kappa + \chi(a)) & & \kappa + \chi(a) < M,\ m < \kappa < M \\
(q, \textsc{Ap}, !, m) \xrightarrow{a} (q', \textsc{Ex}, ?, m + \chi(a)) & & \\
(q, \textsc{Ex}, ?, \kappa) \xrightarrow{a} (q'', \textsc{Ex}, ?, \kappa + \chi(a)) & & q'' = \delta_{\min(\kappa,m)}(q, a),\ m' \leq \kappa < M
\end{array}
$$

For this definition one can verify that $\mathcal{A}'$ computes $T^*_w(q_{\text{in}})$ for the factorization induced by the mode switches. □